ELSEVIER

Discrete Applied Mathematics 93 (1999) 127–140

DISCRETE APPLIED MATHEMATICS

A new algorithm for Jordan sorting: its average-case analysis

E. Sojka *

Department of Computer Science, Technical University of Ostrava, tř. 17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic

Received 29 June 1997; revised 15 June 1998; accepted 22 October 1998

Abstract

Given the intersection points of a planar Jordan curve with the x-axis in the order in which they occur along the curve, sort them into the order in which they occur along the x-axis. In this paper, the average-case analysis of a new simple algorithm that solves the above-mentioned problem is presented. A certain model of generating random Jordan sequences is introduced. The results of the analysis are summarised in the form of theorems that specify the conditions under which the algorithm runs in linear expected time. The results are verified experimentally by a computer simulation. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Jordan sorting; Polygon clipping; Analysis of algorithms

1. Introduction

The term *Jordan curve* is usually used to refer to a closed curve which is a homeomorphic image of a circle. We extend this term to mean either an open or a closed curve. We define the *open Jordan curve* to be a homeomorphic image of a line segment.

Problem 1 (Jordan sorting). *Given the intersection points of a planar Jordan curve with the x-axis in the order in which they occur along the curve, sort them into the order in which they occur along the x-axis* (Fig. 1).

For the input sequence of points in Problem 1, we use the term *Jordan sequence*. For the number (denoted by n) of points in the sequence, we use the term *size of sequence*. In order to make the theoretical analysis simpler, we suppose that the curve crosses the x-axis everywhere it touches it.

* Fax: +420 69 691 9597; e-mail: eduard.sojka@vsb.cz.

PII: S0166-218X(99)00011-6

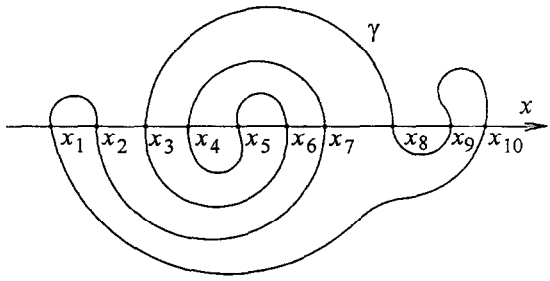

Fig. 1. The x-axis intersects a Jordan curve γ: $x_8, x_3, x_6, x_5, x_4, x_7, x_2, x_1, x_{10}, x_9$ is an example of the input sequence for sorting; $x_1, x_2, \ldots, x_{10}$ is the desired output.

Jordan sorting is involved in polygon and in line clipping, which are basic problems in computer graphics [6,4,1]. Let P be a simple (possibly non-convex) polygon represented by the list of the vertices of its boundary. Let σ be a half-plane (represented by an oriented line). In polygon clipping, the intersection $P \cap \sigma$ is to be found and represented by the lists of the vertices of the resulting simple polygons. In line clipping, a line l is given. The intersection $P \cap l$ is to be found and represented by the endpoints of the resulting line segments. The sorting step involved in the algorithms that solve the above-mentioned problems is equivalent to Problem 1, since we can represent the line parametrically and sort the intersection points according to their values of parameter instead of their x-coordinates. Jordan sorting may also be used in triangulating a simple polygon [8].

We believe that there exist at least the following three reasons why to construct special algorithms for Jordan sorting: (1) They may be faster than the general-purpose algorithms. (2) They may be able to detect whether the input sequence is a Jordan sequence, i.e., they can reveal the erroneous sequences. (3) As a 'side effect', the special algorithms may infer topological information that can also be exploited.

In the algebraic decision-tree model of computation, the worst-case time complexity of Jordan sorting is $\Theta(n)$. The first $\Theta(n)$ worst-case time algorithm was proposed by Hoffmann et al. [3]. Later, this algorithm was simplified by Fung et al. [2]. Unfortunately, in spite of this simplification, both these algorithms are rather complicated, which causes that they are difficult to implement and that they are slow for the inputs of sizes that are of practical interest [7]. Another method of sorting the Jordan sequences is to insert the coordinates of intersection points into the splay tree [5] and then access them in sorted order.

In [7], a new algorithm for Jordan sorting was presented along with the analysis of its worst-case time complexity. (The underlying idea of this algorithm was mentioned in [3], the authors, however, did not develop the idea into the corresponding algorithm and focused on the $\Theta(n)$ worst-case time algorithm.) Although the new algorithm runs in $O(n \log n)$ worst-case time, it was shown that it is useful since the probability of occurrence of the worst-case input is extremely low. Moreover, due to its simplicity, the algorithm can be easily used in practice. In [7], the algorithm was tested and compared

with other known algorithms for Jordan sorting. For the sequences of all tested sizes (from 4 up to 10^4 points) the new algorithm was faster, which was encountered both for the sequences that were hard for the algorithm and for the sequences generated randomly.

This work was motivated by the encouraging results of testing the new algorithm and by our opinion that the asymptotic worst-case time complexity is not a good criterion for judging the practical usefulness of the algorithms for Jordan sorting. The goal of this paper is to carry out the analysis of the new algorithm in an average case. Our main results presented in this paper are summarised in Theorems 3 and 8. The theorems specify the conditions under which the algorithm runs in linear expected time. The results are verified experimentally by a computer simulation.

The paper is organised as follows. In Section 2, the needed terminology is introduced and the algorithm is described. Section 3 is devoted to the analysis of the average-case time complexity of the algorithm. Finally, the experimental results are presented in Section 4.

2. The algorithm

Let γ be a Jordan curve. We suppose that the x-axis intersects γ at n intersection points, denoted by $x_1, x_2, \ldots, x_n$. For presenting the algorithm, the intersection points are numbered in such a way that the sequence $x_1, x_2, \ldots, x_n$ is ordered along the x-axis (Fig. 1). The notation $x_i < x_j$ means that x_i precedes x_j along the x-axis. The intersection points divide the x-axis into intervals, called *segments*. The x-axis cuts the curve into parts, called *arcs*. We use $arc(x_p, x_q)$ to denote the arc whose endpoints are x_p and x_q. The x-axis divides the plane into two half-planes, denoted by U_x and L_x, respectively.

The algorithm constructs successively a planar map, denoted by $M(\gamma)$, that corresponds to the given input of sorting (Fig. 2). In $M(\gamma)$, n nodes correspond to the intersection points $x_1, x_2, \ldots, x_n$; the remaining two nodes x_0 and x_{n+1} are added such that $x_0 < x_1$ and $x_n < x_{n+1}$. The nodes x_0, x_{n+1} are connected by two additional arcs lying in U_x and L_x, respectively (Fig. 2). During sorting, the sequence $M_1, M_2, \ldots, M_n, M(\gamma)$ of the maps is constructed (Fig. 3). The computation starts with the map M_1 containing x_0, x_{n+1}, and the first intersection point from the input sequence. The algorithm then processes the remaining intersection points one by one and updates the map. Once $M(\gamma)$ is found, the ordered sequence $x_1, x_2, \ldots, x_n$ is read from this map.

Suppose that a map M_i has already been created. Let x be the point just being processed, and let $pred(x)$ denote the point that was processed immediately before x (Fig. 4). The process of updating the map M_i into the map M_{i+1} is based on the lemma that follows directly from the fact that the curve does not intersect itself.

Lemma 2. *Let f_r be that face of M_i which is entered by $arc(pred(x), x)$ at $pred(x)$* (Fig. 4). *The point x will fall into one of the segments lying on the boundary of f_r.*

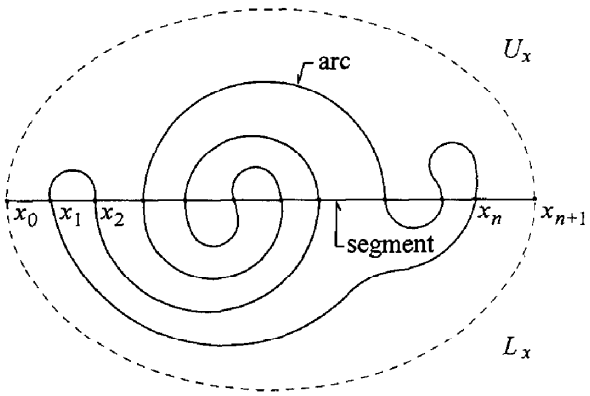

Fig. 2. An example of the map $M(\gamma)$ corresponding to the final stage of sorting.

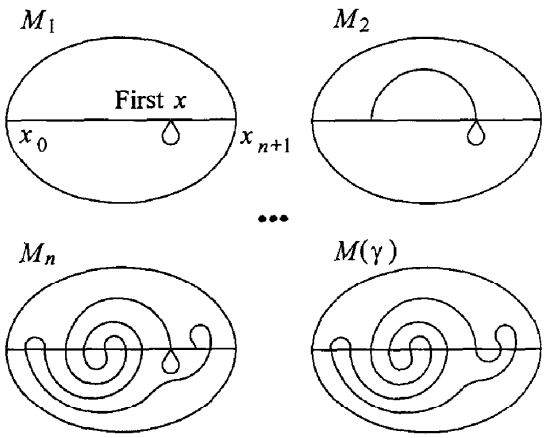

Fig. 3. An example of the sequence of the maps that are constructed during sorting.

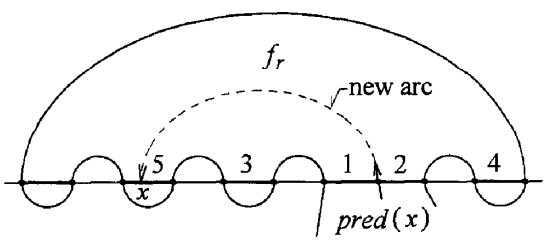

Fig. 4. Illustration of Lemma 2. The size of the face f_r is $size(f_r) = 6$. The numbers show the order in which the segments are tested in the algorithm (see the following text).

In the ith step of the algorithm, the face f_r of the map M_i is split, which gives rise to the map M_{i+1}. This step is referred to as a *transition*. The transition involves: (1) determining the segment containing x and splitting this segment into two segments, (2) splitting the face f_r containing $arc(pred(x),x)$ into two faces f_u, f_v. The faces that arise from splitting f_r are referred to as *descendants* of f_r. Let $size(f_j)$ denote the number of segments lying on the boundary of the face f_j. We use the term *size of face* for this number (Fig. 4). Since each transition increases the number of segments by one, we have

$$size(f_u) + size(f_v) = size(f_r) + 1. \tag{1}$$

If any of the descendants f_u, f_v of the face f_r is chosen for splitting again, its size increases by one. This is due to the fact that during the transition in which a face is chosen for splitting, the segment in which the arc of the curve enters the face is split (Fig. 4). In the algorithm, the maps $M_1, M_2, \ldots, M_n, M(\gamma)$ are represented by a *doubly linked list*. By one pair of pointers, the nodes are connected along the x-axis, by the other pair, they are connected along the curve.

We can now describe the algorithm. Without loss of generality, we can assume that at the first intersection point of the input sequence, the curve passes from L_x to U_x. The algorithm may be formulated as follows (the source code of the algorithm can be found in [7]):

Algorithm 1. (Jordan sorting)

Input. A sequence of the intersection points between a Jordan curve and the x-axis. The points are in the order in which they occur along the curve.

Output. The sequence in which the intersection points are sorted into the order in which they occur along the x-axis.

1. *Read the first intersection point from the input sequence and create the initial map M_1 as depicted in Fig. 3.*

   repeat

2. *Read the next point, denoted by x, from the input. Let f_r denote the face of the map which was entered by $arc(pred(x),x)$ at $pred(x)$; f_r was determined when $pred(x)$ was processed.*
3. *Starting with the segments adjacent to $pred(x)$, test the segments lying on the boundary of f_r sequentially and concurrently in both directions (Fig. 4) whether they contain x. Stop testing if the segment containing x is found.*
4. *Split the segment containing x. Insert the edge representing $arc(pred(x),x)$ into the map.*

   until *all the intersection points from the input are processed;*

5. *Read the output sequence of points from $M(\gamma)$.*

Although the worst-case time complexity of the algorithm is $O(n \log n)$, the analysis presented in [7] showed that the worst running time is achieved only for a special

input. For most inputs, better running times can be expected. This property of the algorithm will be studied in the next section.

3. Time complexity in an average case

In this section, the analysis of the algorithm in an average case will be presented. We will introduce a certain model of generating random Jordan sequences. Then we will analyse the time complexity of the algorithm. The main results are summarised in Theorems 3 and 8.

The time complexity of one transition and the time complexity of the whole algorithm will be measured by the number of tests. By one test we mean the decision whether the point being processed lies inside a segment. Consider a transition that splits a face f_r into two faces f_u, f_v. The time complexity, denoted by t, of this transition is (Fig. 4)

$$t = 2\min\{size(f_u), size(f_v)\}$$

or

$$t = 2\min\{(size(f_u), size(f_v)\} - 1. \tag{2}$$

Note that if the size of one of the faces resulting from splitting (either f_u or f_v) is 1, i.e., if the time complexity of the transition is 1 or 2, then by Eq. (1), the size of the other face is $size(f_r)$, which gives the value of $size(f_r) + 1$ before its next splitting. The transitions with time complexity 1 or 2 thus cause that the sizes of faces may grow.

From the worst-case analysis presented in [7], it follows that the worst running time is achieved only under rather special circumstances consisting in the following: (1) First, a certain number of transitions with time complexity 1 or 2 create 'big' faces in U_x and L_x. (2) In each of the remaining transitions, the biggest face in the appropriate half-plane is chosen and split in half (in [7], we suppose that the biggest face can always be chosen for splitting, which is on the side of worse time complexity).

In Fig. 5, the process of splitting in U_x is illustrated by a tree. In this tree, the inner nodes represent the faces that were split during transitions and replaced by their descendants. The leaves represent the faces that have not been split and that are thus present in the map. The numbers inscribed in the nodes are the sizes of the faces. The size is always measured at the moment when the face has already been chosen for splitting. This rule is also applied to the faces corresponding to the leaves, although, in fact, they have never been chosen.

To study the time complexity of Algorithm 1 for random inputs, we introduce a certain model of generating random Jordan curves and sequences. In this model, the curves are open. The intersection points are generated one by one. Let x denote the point just being generated. By Lemma 2, x must fall into a segment that lies on the boundary of the face containing $arc(pred(x), x)$. This face was determined when $pred(x)$ was generated. In the generator, the segment containing x is chosen randomly.

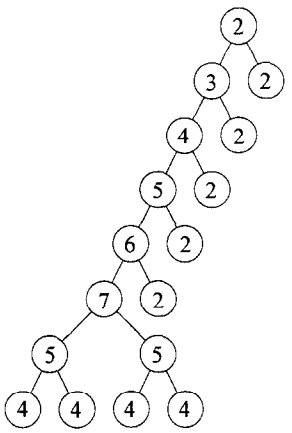

Fig. 5. An example of the tree that depicts the history of splitting in U_x. The tree shows the worst case for $m = n/2 = 8$ ($T = 5 \times 2 + 7 + 2 \times 5 = 27$).

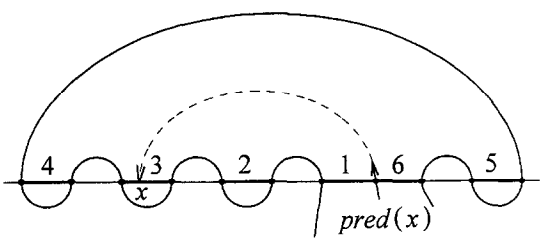

Fig. 6. The numbering of the segments lying on the boundary of a face.

Consider the transitions in U_x. Let m ($m = \lfloor n/2 \rfloor$) denote the number of these transitions. Each face in U_x is a product of a certain sequence of splits. At the beginning of each such sequence, there is the initial face in M_1. Let k denote the number of splits that lie on the path leading from the root to the node representing a certain face (Fig. 5). We say that the face arose at the kth *level of splitting*. At the level $k = 1$, the initial face in M_1 is split into two descendants. The leftmost leaf in Fig. 5, for example, arose in the split whose level was $k = 7$. Clearly, $1 \leqslant k \leqslant m$ for all problems in which m transitions are to be done. Consider a sequence of splits. Let $p_k(s)$ be the probability of the event that just before the split at the kth level, the size of the face is s ($p_{k+1}(s)$ stands for the corresponding probability just after the split has been done). Since the size of the initial face in M_1 is 2, we have $p_1(2) = 1$ and $p_1(s) = 0$ for every $s \neq 2$.

We introduce the numbering of the segments lying on the boundaries of the faces as depicted in Fig. 6 (i.e., we start from the segment to the left of $pred(x)$ and continue along the boundary clockwise). Say that the face in Fig. 6 was split by inserting the new arc. In the analysis, we suppose that the descendant that will be split at the next level of splitting is always that one whose boundary contains the

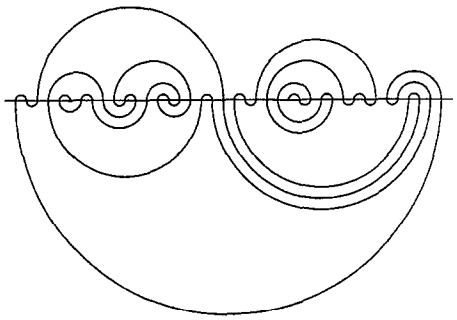

Fig. 7. A curve that was generated randomly. The distribution of probability was $\varphi_k(s,u) = 1/s$.

segment lying to the left of $pred(x)$ which is numbered as a segment 1. (For the descendants lying to the right, we could renumber the segments along the boundary of the face in the opposite direction, which would not influence the results presented in this paper.)

We introduce a function $\varphi_k(s,u)$ expressing the probability of the event that at the kth level of splitting and in a face of size s, the intersection point will fall into the segment whose number is u. For all possible values of k, we have a set $\{\varphi_k(s,u) \mid 1 \leqslant k \leqslant m\}$ of such distributions of probability. The choice of the distributions in $\{\varphi_k(s,u)\}$ influences the 'appearance' of the curves produced by the generator. In this way, the generator can even be forced to produce the worst-case curves. Fig. 7 shows an example of the curve that was created by the generator. In this example, we set $\varphi_k(s,u) = 1/s$ for all k, i.e., all the segments lying on the boundaries of the faces always had equal probabilities that they would contain x. Since it is unclear what should be regarded as a practical average Jordan curve and Jordan sequence, we studied the conditions that guarantee the linear expected running time of the algorithm. Obviously, if these conditions are reasonable, i.e., not too restrictive, the algorithm is useful.

From now on, by the size of face we mean the size that is measured at the moment when the face has already been chosen for splitting. Thus, the smallest face we work with is a face of size 2. The biggest face that can occur after m transitions in U_x (and after the same number of transitions in L_x) is a face of size $m+2$. The descendant of size s_d can occur in such a way that in a face whose size is $s \geqslant s_d - 1$, the intersection point x will fall into the segment whose number is $s_d - 1$. This gives the following equations:

$$p_{k+1}(2) = \varphi_k(2,1)p_k(2) + \varphi_k(3,1)p_k(3) + \varphi_k(4,1)p_k(4) + \cdots + \varphi_k(m+2,1)p_k(m+2),$$

$$p_{k+1}(3) = \varphi_k(2,2)p_k(2) + \varphi_k(3,2)p_k(3) + \varphi_k(4,2)p_k(4) + \cdots + \varphi_k(m+2,2)p_k(m+2),$$

$$
\begin{aligned}
p_{k+1}(4) &= \varphi_k(3,3)p_k(3) + \varphi_k(4,3)p_k(4) \\
&\quad + \cdots + \varphi_k(m+2,3)p_k(m+2), \\
&\vdots \\
p_{k+1}(m+2) &= \varphi_k(m+1,m+1)p_k(m+1) \\
&\quad + \varphi_k(m+2,m+1)p_k(m+2).
\end{aligned} \tag{3}
$$

(It is worth mentioning that the process of splitting can be viewed as a Markov process.) The probabilities $p_k(2), p_k(3), p_k(4), \ldots, p_k(m+2)$ can be arranged into the vector $\boldsymbol{p}_k = (p_k(2), p_k(3), p_k(4), \ldots, p_k(m+2))^{\mathrm{T}}$ ($^{\mathrm{T}}$ indicates the transposition). Since the size of the initial face in M_1 is 2, we have $\boldsymbol{p}_1 = (1,0,0,\ldots,0)^{\mathrm{T}}$. Similarly, the values of the function $\varphi_k(s,u)$ can be arranged into a square matrix, denoted by Φ_k, which is of the form

$$
\Phi_k = \begin{bmatrix}
\varphi_k(2,1) & \varphi_k(3,1) & \varphi_k(4,1) & \varphi_k(5,1) & \varphi_k(6,1) & \ldots \\
\varphi_k(2,2) & \varphi_k(3,2) & \varphi_k(4,2) & \varphi_k(5,2) & \varphi_k(6,2) & \ldots \\
0 & \varphi_k(3,3) & \varphi_k(4,3) & \varphi_k(5,3) & \varphi_k(6,3) & \ldots \\
0 & 0 & \varphi_k(4,4) & \varphi_k(5,4) & \varphi_k(6,4) & \ldots \\
0 & 0 & 0 & \varphi_k(5,5) & \varphi_k(6,5) & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots
\end{bmatrix}. \tag{4}
$$

Rewriting Eq. (3) into the matrix form and applying the result recursively, we obtain

$$
\boldsymbol{p}_{k+1} = \Phi_k \boldsymbol{p}_k = \Phi_k \Phi_{k-1} \cdots \Phi_1 \boldsymbol{p}_1 = \left(\prod_{i=1}^{k} \Phi_i \right) \boldsymbol{p}_1. \tag{5}
$$

Note that the dimensions of the vectors and matrices may be greater than $m+1$, which is the value considered up to now. If m transitions are to be carried out, no face of size greater than $m+2$ occurs. This fact is taken into account by zero probabilities of existence of the faces whose size is greater than the maximal possible value following from the number of transitions that were carried out. The size $Q+1$ thus can be used for all $m \leqslant Q$.

Let $\mu_{s,k}$ denote the mean value of the size of face just before the kth level of splitting. Since the size of the initial face in M_1 is 2, we have $\mu_{s,1} = 2$. We introduce the vector $\boldsymbol{s} = (2,3,\ldots,Q+2)^{\mathrm{T}}$. From the definition of mean value and from Eq. (5), we obtain

$$
\mu_{s,k} = \sum_{s=2}^{Q+2} s p_k(s) = \boldsymbol{s}^{\mathrm{T}} \boldsymbol{p}_k = \boldsymbol{s}^{\mathrm{T}} \left(\prod_{i=1}^{k-1} \Phi_i \right) \boldsymbol{p}_1. \tag{6}
$$

We introduce a function $\tau_k(s,t)$ expressing the probability of the event that at the kth level of splitting, the time complexity of one transition in a face of size s is just t. From the order in which the segments are numbered (Fig. 6) and from the order in which the segments are tested in the algorithm (Fig. 4), it is clear that $\tau_k(s,1)=\varphi_k(s,1)$, $\tau_k(s,2)=\varphi_k(s,s)$, $\tau_k(s,3)=\varphi_k(s,2)$, $\tau_k(s,4)=\varphi_k(s,s-1)$, etc. Obviously, $\tau_k(s,t)=0$ whenever $s<t$, since the transition with time complexity t can be carried out only in a face whose size is at least t. Let $q_k(t)$ be the probability of the event that at the kth level of splitting, the time complexity of one transition is just t. We have

$$q_k(t)=\sum_{s=2}^{Q+2}\tau_k(s,t)p_k(s). \tag{7}$$

We use $\mu_{t,k}$ to denote the mean value of the time complexity of one transition at the kth level of splitting. It follows that

$$\mu_{t,k}=\sum_{t=1}^{Q+1} tq_k(t). \tag{8}$$

Note that although we expect the faces of size up to $Q+2$, the face of size $Q+2$ is never split. The biggest face that can be split is a face of size $Q+1$. This explains the upper bound of $Q+1$ in the previous sum. We introduce the vectors $\boldsymbol{q}_k=(q_k(1),q_k(2),\ldots,q_k(Q+1))^{\mathrm{T}}$, $\boldsymbol{t}=(1,2,3,\ldots,Q+1)^{\mathrm{T}}$ and the matrix

$$\boldsymbol{T}_k=\begin{bmatrix}
\varphi_k(2,1) & \varphi_k(3,1) & \varphi_k(4,1) & \varphi_k(5,1) & \varphi_k(6,1) & \ldots\\
\varphi_k(2,2) & \varphi_k(3,3) & \varphi_k(4,4) & \varphi_k(5,5) & \varphi_k(6,6) & \ldots\\
0 & \varphi_k(3,2) & \varphi_k(4,2) & \varphi_k(5,2) & \varphi_k(6,2) & \ldots\\
0 & 0 & \varphi_k(4,3) & \varphi_k(5,4) & \varphi_k(6,5) & \ldots\\
0 & 0 & 0 & \varphi_k(5,3) & \varphi_k(6,3) & \ldots\\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots
\end{bmatrix}. \tag{9}$$

Eqs. (7) and (8) can be rewritten into the matrix form

$$\boldsymbol{q}_k=\boldsymbol{T}_k\boldsymbol{p}_k, \tag{10}$$

$$\mu_{t,k}=\boldsymbol{t}^{\mathrm{T}}\boldsymbol{q}_k=\boldsymbol{t}^{\mathrm{T}}\boldsymbol{T}_k\boldsymbol{p}_k=\boldsymbol{t}^{\mathrm{T}}\boldsymbol{T}_k\left(\prod_{i=1}^{k-1}\boldsymbol{\Phi}_i\right)\boldsymbol{p}_1. \tag{11}$$

Furthermore, we introduce

$$\mu_{s,\max}=\max_k\{\mu_{s,k}\},\quad \mu_{t,\max}=\max_k\{\mu_{t,k}\}. \tag{12}$$

Obviously, if the value $\mu_{t,\max}$ is smaller than a constant independent of the size of the problem, then Algorithm 1 runs in linear expected time that does not exceed $n\mu_{t,\max}$. (Note that the sequence $\mu_{t,1},\mu_{t,2},\mu_{t,3},\ldots$ is not necessarily required to converge; the requirement of convergence is stronger.)

In the following theorem, we study a special case in which we assume that at all levels of splitting and in the faces of all sizes, all the segments lying on the boundary of one face always have equal probabilities that they will contain the new intersection point.

Theorem 3. *Let each distribution of probability in* $\{\varphi_k(s,u)\}$ *be* $\varphi_k(s,u) = 1/s$, *then Algorithm* 1 *runs in* $T \leqslant 2n$ *expected time.*

Proof. By Eqs. (4) and (9), the matrices Φ_k and T_k are of the form

$$\Phi_k = T_k = \begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \cdots \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \cdots \\ 0 & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \cdots \\ 0 & 0 & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \cdots \\ 0 & 0 & 0 & \frac{1}{5} & \frac{1}{6} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}. \tag{13}$$

Applying Eq. (11), we obtain

$$\mu_{t,k} = 2 - \frac{1}{2^k}. \tag{14}$$

Note that the values $\mu_{t,1}, \mu_{t,2}, \mu_{t,3}, \ldots$ do not depend on the size of the problem. The size is taken into account by considering the appropriate number of terms when determining the maximum in Eq. (12). It is easily seen that for $k \to \infty$, the value of $\mu_{t,k}$ converges to 2, which is the maximal possible mean value of the time complexity of one transition. The theorem follows. □

In the sequel, we will show that the good performance of the algorithm can also be expected under less restrictive assumptions.

Definition 4. We call the distribution $\varphi_k(s,u)$ of probability *symmetric* if $\varphi_k(s,u) = \varphi_k(s,s-u+1)$ for every s and for every $u \in \{1,2,\ldots,s\}$.

Definition 5. In a face, we define the *distance* of a segment g from $pred(x)$ to be the minimal number of segments that are visited when walking from $pred(x)$ along the boundary of the face (in any of both possible directions) to the segment g (Fig. 8).

It is easily seen that in every symmetric distribution, the probability of the event that a segment will contain the new intersection point depends on the distance of the segment from $pred(x)$.

Lemma 6. *Let all the distributions of probability in* $\{\varphi_k(s,u)\}$ *be symmetric, then* $\mu_{s,k} \leqslant 3$ *for all* k.

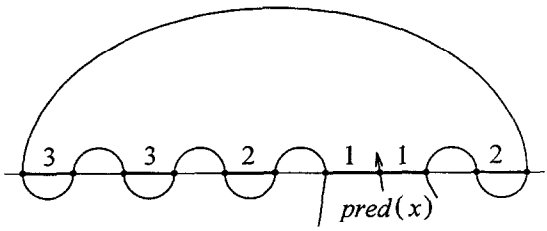

Fig. 8. Illustration of Definition 5. The numbers show the distances of the particular segments from $pred(x)$.

Proof. We introduce the difference $\Delta\mu_{s,k,k+1} = \mu_{s,k+1} - \mu_{s,k}$. Furthermore, we introduce the value $\mu_{u,k}(s)$ as follows:

$$\mu_{u,k}(s) = \sum_{u=1}^{s} u\varphi_k(s,u). \tag{15}$$

From Eqs. (5) and (6), we have

$$\Delta\mu_{s,k,k+1} = \mu_{s,k+1} - \mu_{s,k} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{\Phi}_k\boldsymbol{p}_k - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{p}_k = \boldsymbol{s}^{\mathrm{T}}(\boldsymbol{\Phi}_k - \boldsymbol{I})\boldsymbol{p}_k. \tag{16}$$

Substituting Eq. (4) into Eq. (16) and applying Eq. (15), we obtain

$$\Delta\mu_{s,k,k+1} = \sum_{s=2}^{Q+2} [\mu_{u,k}(s) - s + 1)]p_k(s). \tag{17}$$

It is easy to check that $\mu_{u,k}(s) = (s+1)/2$ for every symmetric distribution. Therefore,

$$\Delta\mu_{s,k,k+1} = \tfrac{1}{2}(3 - \mu_{s,k}). \tag{18}$$

Finally, since $\mu_{s,1} = 2$, we conclude from Eq. (18) that

$$\mu_{s,k} = 3 - \frac{1}{2^{k-1}}, \tag{19}$$

which proves the lemma. □

Lemma 7. *For every set* $\{\varphi_k(s,u)\}$ *of distributions of probability, the inequality* $\mu_{t,k} \leqslant \mu_{s,k}$ *holds for all* k.

Proof. Combining Eq. (8) with Eq. (7), we obtain

$$\mu_{t,k} = \sum_{t=1}^{Q+1} t \sum_{s=2}^{Q+2} \tau_k(s,t)p_k(s) = \sum_{s=2}^{Q+2} p_k(s) \sum_{t=1}^{Q+1} t\tau_k(s,t). \tag{20}$$

Since $\sum_{t=1}^{Q+1} t\tau_k(s,t)$ is the mean value of the time complexity of one transition in a face of size s and since this value is no greater than s, we conclude that

$$\mu_{t,k} \leqslant \sum_{s=2}^{Q+2} s\,p_k(s) = \mu_{s,k}, \tag{21}$$

which completes the proof. □

Table 1
The mean values and the standard deviations of the number of tests per one transition for $\varphi_k(s,u) = 1/s$

Size of problem (n)	10	100	1000	10 000	100 000
Number of experiments	10 000	10 000	1000	1000	500
E[Number of tests/n]	1.70	1.97	1.99	2.00	2.00
$\sqrt{\text{Var}}$[Number of tests/n]	0.28	0.12	0.04	0.012	0.004

Theorem 8. *Let all the distributions of probability in* $\{\varphi_k(s,u)\}$ *be symmetric, then Algorithm* 1 *runs in* $T \leqslant 3n$ *expected time.*

Proof. By Lemma 7, the mean value of the time complexity of one transition is no greater than $\max\{\mu_{s,k}\}$. By Lemma 6, $\mu_{s,k} \leqslant 3$ for all k. The theorem follows. □

4. Experimental results

We implemented the generator described in Section 3. We set $\varphi_k(s,u) = 1/s$ for all k. The sequences created by the generator were sorted by Algorithm 1. We evaluated the mean value and the standard deviation of the number of tests per one transition. The decision whether the intersection point lies inside a segment was considered to be one test. The results are summarised in Table 1.

5. Conclusion

We have presented the average-case analysis of the algorithm for Jordan sorting that was published in [7] and briefly described also in this paper. Our effort was motivated by the encouraging results of testing the algorithm and comparing the algorithm with other known algorithms [7], by the fact that, due to its simplicity, the algorithm can be easily used in practice, and by our opinion that the asymptotic worst-case time complexity is not a good criterion for judging the practical usefulness of the algorithms for Jordan sorting. We note that after a minor modification, the algorithm discussed here can also detect whether the input sequence is a Jordan sequence. The map which is created during sorting can be used, for example, in polygon clipping.

The goal of this paper was to introduce a certain model of generating random Jordan sequences and to analyse the algorithm. The results of the analysis are summarised in Theorems 3 and 8. The theorems show that for an interesting class of Jordan sequences, the algorithm runs in linear expected time. Let us also point out that the theorems do not seem to exhaust all the possible cases in which the linear expected running time can be achieved. We leave for future work to study the remaining cases.

References

[1] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, Computer Graphics: Principles and Practice, Addison-Wesley, Reading, MA, 1990.
[2] K.Y. Fung, T.M. Nicholl, R.E. Tarjan, C.J. van Wyk, Simplified linear-time Jordan sorting and polygon clipping, Inform. Process. Lett. 35 (1990) 85–92.
[3] K. Hoffmann, K. Mehlhorn, P. Rosenstiehl, R.E. Tarjan, Sorting Jordan sequences using level-linked search trees, Inform. Control 68 (1986) 170–184.
[4] Y.D. Liang, B.A. Barsky, An analysis and algorithm for polygon clipping, Commun. ACM 26 (1983) 868–877.
[5] D.D. Sleator, R.E. Tarjan, Self-adjusting binary search trees, J. Assoc. Comput. Mach. 32 (1985) 652–686.
[6] I.E. Shutherland, G.W. Hodgman, Reentrant polygon clipping, Commun. ACM 17 (1974) 32–42.
[7] E. Sojka, Two simple and efficient algorithms for Jordan sorting and polygon cutting and clipping, Comput. Networks ISDN Systems 29 (1997) 1661–1673.
[8] R.E. Tarjan, C.J. van Wyk, An $O(n \log \log n)$ time algorithm for triangulating a simple polygon, SIAM J. Comput. 17 (1988) 143–178; Erratum: SIAM J. Comput. 17 (1988) 1061.